ENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1993	Professional Paper

4. TITLE AND SUBTITLE

EVOLVING NEURAL NETWORK ARCHITECTURE

5. FUNDING NUMBERS

In House Funding

6. AUTHOR(S)

J. R. McDonnell, D. Waagen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division
San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division
San Diego, CA 92152–5001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This work investigates the application of a stochastic search technique, evolutionary programming, for developing self-organizing neural networks. The chosen stochastic search method is capable of simultaneously evolving both network architecture and weights. The number of synapses and neurons are incorporated into an objective function so that network parameter optimization is done with respect to computational costs as well as mean pattern error. Experiments are conducted using feedforward networks for simple binary mapping problems.

Published in *Proceedings*, SPIE 92 International Symposium Neural and Stochastic Methods in Image and Signal Processing, Vol 1766.

14. SUBJECT TERMS		15. NUMBER OF PAGES
neural networks	mean pattern error	
stochastic search technique		16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a NAME OF RESPONSIBLE INDIVIDUAL	21b TELEPHONE *(Include Area Code)*	21c OFFICE SYMBOL
J. R. McDonnell	(619) 553–5762	Code 731

Evolving neural network architecture

John R. McDonnell and Don Waagen

NCCOSC, RDT&E Div.
San Diego, CA 92152-5000

ABSTRACT

This work investigates the application of a stochastic search technique, evolutionary programming, for developing self-organizing neural networks. The chosen stochastic search method is capable of simultaneously evolving both network architecture and weights. The number of synapses and neurons are incorporated into an objective function so that network parameter optimization is done with respect to computational costs as well as mean pattern error. Experiments are conducted using feedforward networks for simple binary mapping problems.

1. INTRODUCTION

The neural network design process is largely heuristic. The designer's previous experience (or the work of other researchers) oftentimes dictates an initial network configuration for the problem at hand. If the network can be trained to achieve the designer's goals, then the design process is terminated. If success is not attained, then a testing phase which is largely trial and error ensues. The end result can often be a network with excess parameters and little regard for computational costs.

This work attempts to incorporate the computational costs associated with neural network configurations into the optimization procedure. Potential benefits of a network which has an optimized architecture include increased throughput for real-time signal processing applications as well as decreased memory requirements. Determining both network parameters and structure simultaneously requires a search procedure which is amenable to combinatorial optimization tasks. The more successful algorithms for these types of problems have generally been stochastic such as simulated annealing, genetic algorithms and simulated evolution. The simulated evolution[1], or evolutionary programming (EP), paradigm has been shown to have the desired attributes: combinatorial optimization capabilities[2], the ability to train neural networks[3] as well as determine model structure[4].

Many investigations have been undertaken in an effort to obtain desired I/O mappings with variable configuration networks. Ash[5] developed the dynamic node creation (DNC) algorithm which created new nodes in the hidden layer when the training error rate fell below an arbitrarily chosen critical value. Hirose *et al.*[6] use the same approach for node creation, but also remove nodes when small error values are attained. Dow and Sietsma[7,8] have developed post-processing algorithms which remove or prune excess nodes from network hidden layers. These algorithms are implemented as a two stage process. The first stage removes nodes which have constant outputs or are complimentary to other nodes. The second stage removes nodes that are independent of the other nodes within the layer and whose output is not required to achieve a solution.

Li[9] has developed a generalization of the backpropagation algorithm which minimizes both the total system error and the norm of the weights. In simulation, this approach yielded null portions of the weight space providing an implicit means of network optimization (weights and structure). Simulation results also resulted in nearly invariant weight sets regardless of the initial weight values whereas straight backpropagation did not. Bailey[10] generates neural network structure by evaluating synapse importance and system error (synapse effectiveness) criteria. If both synapse effectiveness and importance become low then it becomes eligible for removal. Tenorio and Lee[11] have applied simulated annealing to a group method data handling (GMDH) scheme to obtain a self-organizing neural network (SONN). A hierarchical minimum description length (MDL) function called the structure estimation criteria is used as the objective function. The resulting network is sparsely connected as each node is arbitrarily limited to two inputs. Since a low order Kolmogorov-Gabor polynomial approximation is implemented for the node activation function, more weights than connections exist. Hertz *et al.*[12] give a concise overview of other work on neural network construction and pruning algorithms.

The premise of this work is that the designer can postulate an objective function which, if optimized, will yield desirable results for the task at hand. Further, the approach used in this study seeks to take advantage of computational resources during the design/training phase thereby removing the burden of evaluation by trial-and-error from the designer. For purposes of discussion, Fig. 1 illustrates the structure of a hypothetically evolved neural network. The connectivity and number of nodes in the hidden layer are determined via a multi-agent stochastic search technique. The application of EP to this task is investigated in three distinct phases. Initially, the EP training algorithm for neural networks given in the next section is generalized so that connectivity between nodes can be randomly selected. In the next phase of the study, the number of hidden units is a random variable which is incorporated into the stochastic search. The final phase integrates methods employed in the previous phases into a single EP search algorithm to achieve a network such as that shown in Fig. 1.

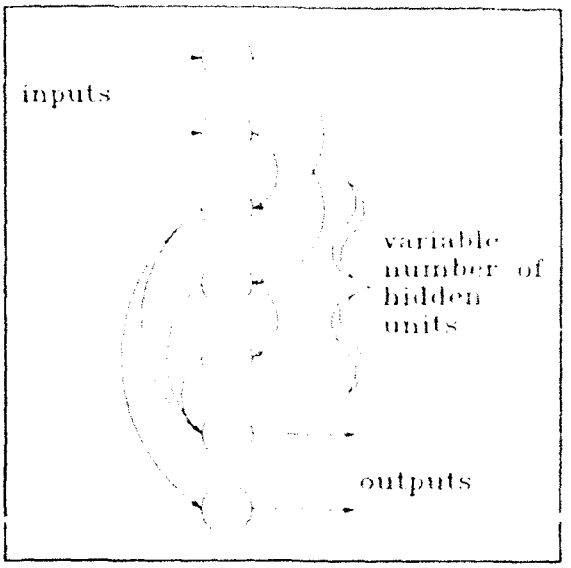

Figure 1. A hypothetically evolved network structure with variable hidden units and connectivity.

The evolutionary programming paradigm is outlined in the next section along with applications of the EP search technique to fully-connected feedforward neural networks. Subsequent sections investigate evolving the connectivity structure of the network and the number of nodes in the hidden layer(s). These methods are then combined to yield self-organizing neural networks.

2. APPLYING EVOLUTIONARY PROGRAMMING TO NEURAL NETS

2.1. Evolutionary Programming

Evolutionary programming is a neo-Darwinian search paradigm first suggested by Fogel *et al.*[1] This stochastic search method is typically applied as a global optimizer. EP has been successfully applied to a variety of optimization problems including the traveling salesman problem, parameter estimation and system ID, and neural net training.

The EP optimization algorithm for locating extrema of any general function can be described by the following steps:

1. *Form an initial population $P_{2N-1}(x)$ of size 2N. The parameters x associated with parent element P_i are randomly initialized from a user specified search domain.*
2. *Assign a fitness score $S_i(x)$ to each element $P_i(x)$ in the population.*
3. *Reorder the population based on the number of wins generated from a stochastic competition process.*
4. *Generate offspring (P_N .... P_{2N-1}) of the highest ranked N elements (P_0 .... P_{N-1}) in the population.*
5. *Loop to step 2.*

The generality of this algorithm lends power to its implementation besides providing a systematic means of stochastic search. The user is not bound to any particular coding structure nor mutation strategy. As previously stated, EP is used in this investigation since it is well suited for simultaneously evolving model structure and parameters.

2.2. Training Neural Networks with EP

2.2.1. Determining Network Weights

Since EP is a general purpose optimization procedure, it can be readily used for determining neural network connection strengths. The objective function of the EP training approach for fully-connected feedforward networks is the same as that used in backpropagation: minimize the sum-squared error function $E = \frac{1}{2}\sum_p\sum_k(t_k - o_k)^2$. A common metric is to normalize

E by the number of training patterns presented. This section applies the algorithm given in the previous section to neural networks and then shows results for sample training runs using simple binary mappings. It should be noted that the examples given are by no means statistically significant and are primarily shown for exemplary purposes.

Each fully-connected feedforward network Φ_i in the population P of networks may be represented by the weight array

$$\Phi_i = w[i][layer][from\_node][to\_node]$$

For this application, the initial population was instantiated with weights chosen from a $U[-0.5, 0.5]$ distribution. Next, a cost is assigned to each network in the configuration. The N members of the population generate offspring (perturbed weight sets) according the equation

$$W_o = W_p + \delta W_p$$

where typically $\delta W_p \in N(0, S_F \cdot E)$ with a scaling coefficient S_F. The scaling factor is a probabilistic analog to the stepsize used in gradient descent methods and may also be treated as a random variable within the EP search strategy. The variance of the weight perturbations is bound by the total system error in this application. To emulate the probabilistic nature of survival, a pairwise competition is held where individual elements compete against randomly chosen members of the population. For example, if network Φ_j is randomly selected to compete against network Φ_i, then a win is awarded to network Φ_i if $E_i < E_j$. Another approach would be to let the probability of network Φ_i winning be $E_j/(E_i+E_j)$. The N networks with the most wins are then used to generate offspring and the process is repeated.

Figures 2-4 shows the effect of varying different parameters in training a 2-2-1 neural net for the XOR mapping. Fig. 2 shows the best mean sum-squared pattern error for various scaling factors from a population with 10 parent networks. As shown, modifying the scaling factor effects the convergence rate of the training. The smaller scaling factors tend to produce smoother curves (smaller changes) while the larger scaling factors yield more dramatic results due to larger step sizes. The number of networks over which the search is conducted is generally user defined. However, the population size need not be a static parameter as it can increase or decrease over the generations of the search. Training results with a 5x increase in population size is shown in Fig. 3 for a scaling factor of $S_F = 100$. Fig. 4 shows successful training is possible with a randomly chosen scaling factor for $S_F \in U[0,1000]$ and $N = 10$. The training run shown did not converge as fast as other runs, but it did produce a smoother scaling factor sequence.

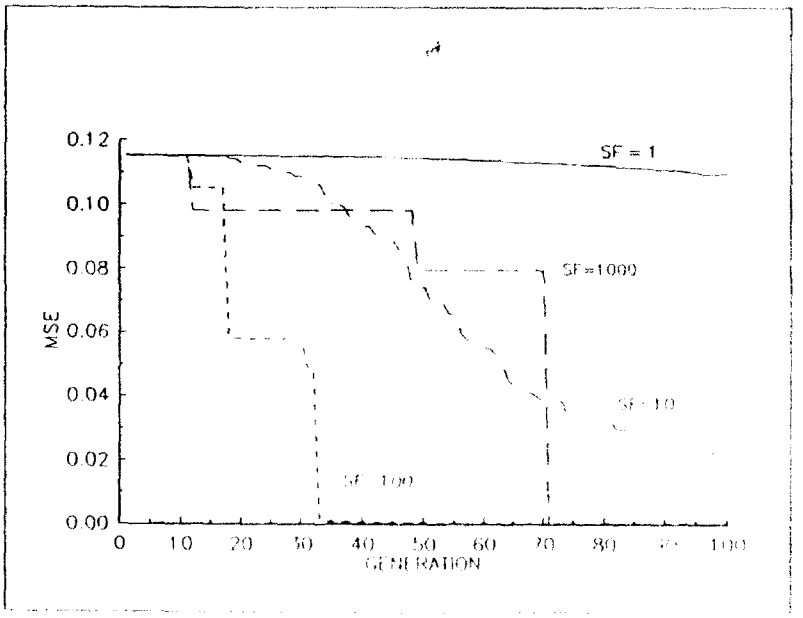

Figure 2. EP training of a 2-2-1 XOR mapping network for various scaling factors, N = 10.

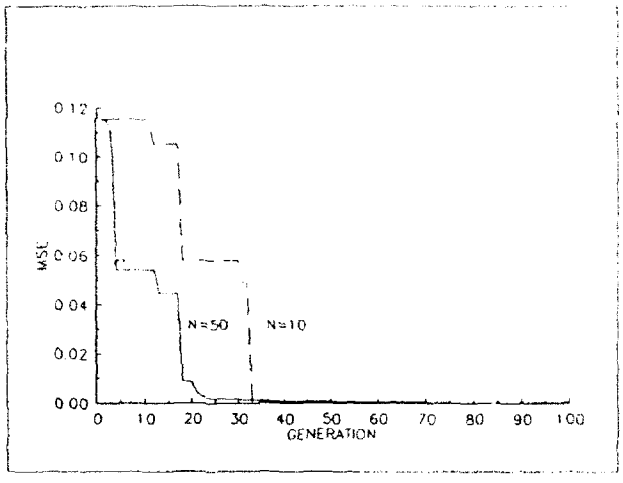

Figure 3. EP training of XOR mapping network with different population sizes, $S_F = 100$.

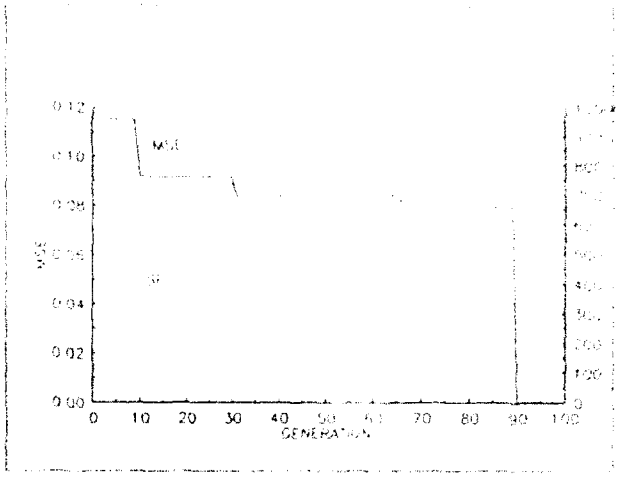

Figure 4. EP training of XOR mapping network with variable S_F.

2.2.2. Hard-limited Activations

Besides yielding only locally optimal solutions, it is necessary that activation functions be continuously differentiable when gradient descent methods are employed for training multi-layer perceptrons (MLPs). The continuity condition is not a requirement when stochastic methods such as simulated annealing[13], genetic algorithms[14] or EP are used. An interesting method has been developed by Winter and Widrow[15] for training MLPs with bipolar activation functions. This method, termed the Madaline Rule II, consists of applying the minimal disturbance principle to weights associated with individual ADALINES. If better results are obtained, then the new weight values are kept; otherwise, the new weights are ignored. If the training process exhausts trials involving a single ADALINE, pairwise (or higher) adaptations are attempted.

The 3-bit parity problem has been chosen as an example for training MLPs with hard-limited activation functions using EP. Fig. 5 compares training results when sigmoidal and binary activation functions and are used in a *3-3-1* network. As shown, the discrete system error states which result when using a binary activation function are quite pronounced. The concept of a different activation function for different neurons can also be easily employed within an EP search. Each different activation function could be treated as a random variable and be discretely indexed within a neuron structure.

2.2.3. Combining Backpropagation and EP

It is possible to do a local and global search in parallel. Such a strategy has been investigated by Waagen *et al.*[16] in the development of the Stochastic Direction Set (SDS) method. This method maintains the integrity of searching for a solution set of parameters without explicitly requiring the analytical function or empirically determined derivatives. The SDS method uses an augmented (with orthogonal decomposition) Hooke-Jeeves technique for the local search while EP is used for finding the global solution neighborhood. The variance of the stochastic step size is held constant over the duration of the search. Two offspring are generated for each base point: one locally and one globally. A local offspring replaces its parent if it achieves a better fitness score. EP and backpropagation training can be implemented together in a similar parallel fashion. Fig. 6 shows a serial implementation wherein a *3-3-1* network is first trained with EP and then refined using backpropagation for the 3-bit parity mapping. This sample run was generated with 10 parent networks and a scaling factor $S_F = 3.3$. The backpropagation parameters included a stepsize of $\eta = 0.9$ and a momentum coefficient of $\alpha = 0.2$.

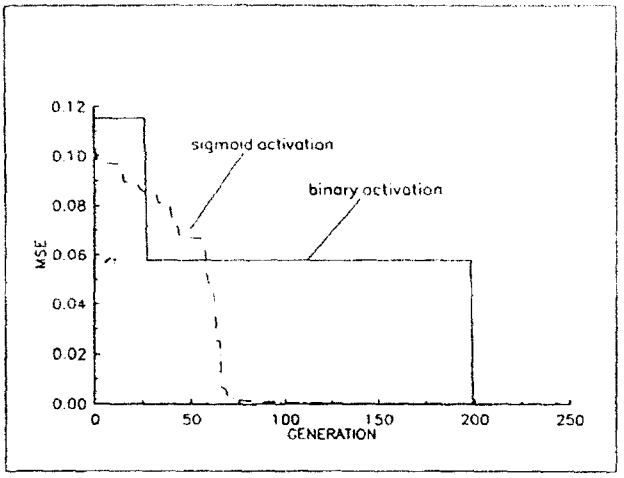

Figure 5. EP training for 3-bit parity mapping with both sigmoidal and binary activation functions.

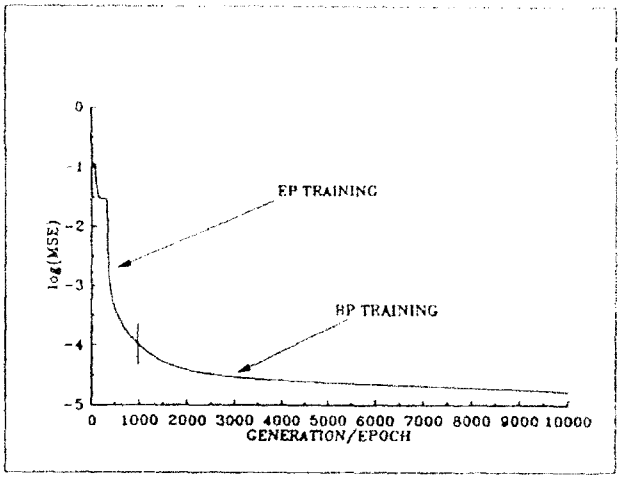

Figure 6. A serial EP-BP training session for the 3-bit parity mapping.

3. EVOLVING CONNECTIVITY

This portion of the investigation addresses reducing the number of synapses between the neurons. Typically, work in this area incorporates a function of the weight magnitudes in the objective function. Hertz *et al.*[12], for example, discuss optimizing an objective function J described by

$$J = E + \frac{1}{2}\gamma\sum_{ij}\frac{w_{ij}^2}{1 + w_{ij}^2}$$

where E is the mean sum-squared pattern error. Weigend *et al.*[17] implement a similar modification with the weights normalized by an arbitrary parameter w_o

$$J = E + \gamma\sum_{ij}\frac{(w_{ij}/w_o)^2}{1 + (w_{ij}/w_o)^2}$$

This study investigates modifying the object function according to

$$J = \alpha E + \beta N_{connections}$$

whereby the cost associated with the number of connections, $N_{connections}$, between neurons is scaled and incorporated into the cost function. For these studies a value of $\beta = 0.0001$ was used with $\alpha = 1$. A heuristic which might be employed would be to let $\beta \approx \alpha E_{crit}/N_{max}$ thereby incorporating the desired training error and the maximum number of connections possible over the specified domain to reasonably weight the cost associated with the number of connections.

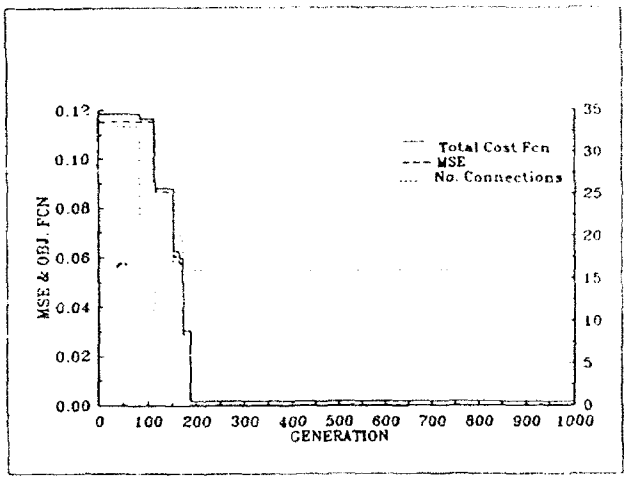

Figure 7. Evolving connectivity for the XOR mapping. The final architecture is shown to the right.

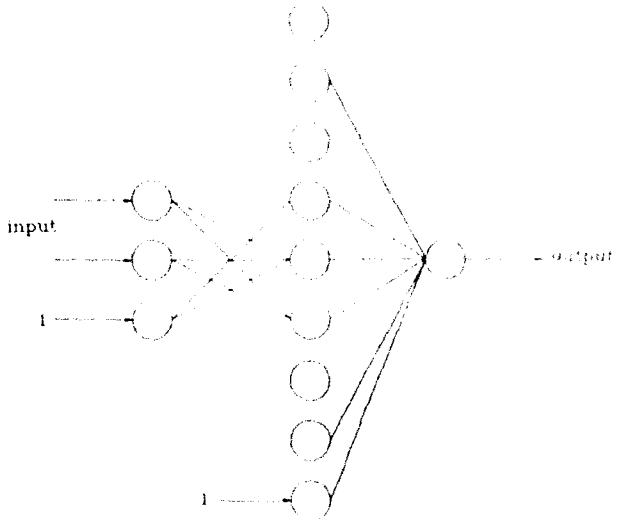

Figure 8. The final evolved connectivity for the XOR mapping network.

To implement this capability, a connectivity array has been utilized where *c[i][layer][from_node][to_node]* = *1* if a connection exists and 0 if no connection is present. At the start of the run, the connectivity array is set to 1 giving a fully-connected feedforward network. The designer must specify the number of neurons over which the search is conducted. This determines the maximum number of connections. From the range of connections, a synapse is chosen at random and modified based on its current state. It is connected if it has been disconnected or disconnected if it is currently connected. Essentially, the bit is flipped. The number of connections which may be affected at each mutation is arbitrarily set by the designer or determined in a random fashion. The connection array is incorporated in the neuron input dot product term thereby nulling any signals between disconnected neurons. As before, weights are continually modified in the event that a neuron pair is reconnected.

The results of evolving connectivity using a network with 8 hidden units for the XOR mapping is shown in Figures 7 and 8. After 1000 generations, the network with the lowest cost function has 12 connections as shown in Fig. 8. Although the structure can be reduced to replicate a fully-connected 2-2-*1* feedforward architecture, it must be done through post-processing.

In an effort to place greater emphasis on signal propagation through the network, an alternative strategy has been developed though not yet implemented. This strategy assigns a probability of connection from N_{jk}, neuron k in layer j, to subsequent layers contingent on inputs to all the neurons in level j

$$p_c(N_{jk}) = \frac{\sum_i o_i w_{ik}}{\sum_j \sum_i o_i w_{ij}}$$

This technique assumes a random connection process exists between the input layer and first hidden layer. Poor solutions (such as excess emphasis on a single node) will not be sustained unless the fitness value is competitive with other member of the population.

4. EVOLVING HIDDEN NODES

The standard backpropagation approach to training neural networks entails specifying the number of layers and the number of nodes in each layer. This portion of the investigation allows the number of nodes in the hidden layer(s) to be treated as a random variable. The EP training technique given in section 2.2.1 is still used although other training methods (such as backpropagation) may be applicable.

The method outlined in section 2.2.1 is slightly modified to allow for a variable number of hidden nodes. The maximum number of hidden nodes specified in the program is the domain over which the search is conducted. This has the obvious shortcoming that it is still possible to under estimate the number of nodes which may be required to solve a particular problem. This portion of the investigation maintains a fully-connected feed-forward architecture. The cost function used is given by[7]

$$J = \alpha E + \gamma N_{nodes}$$

where $\alpha = 1$ and $\gamma = 0.01$. A scaling factor of $\gamma = 0.01$ allows smaller networks with larger mean sum-squared pattern error terms to be selected. At each weight perturbation, the number of hidden nodes is randomly selected from a user specified domain.

Fig. 9 shows the results for the XOR mapping problem with varying hidden units. The maximum number of hidden units possible was arbitrarily set at 20. After 200 generations the network has evolved a *2-2-1* structure as well as an appropriate set of weight parameters. At the end of 500 generations a mean sum-squared pattern error of 0.00003 is attained. Fig. 10 shows the results for the XOR mapping problem if the search domain is increased to 50 hidden units. In this case, a *2-2-1* structure is also evolved simultaneously with the weight parameters. At 500 generations a mean sum-squared pattern error of 0.0001 is achieved. Both sets of experiments were conducted with $S_P = 100$ and 10 parents.

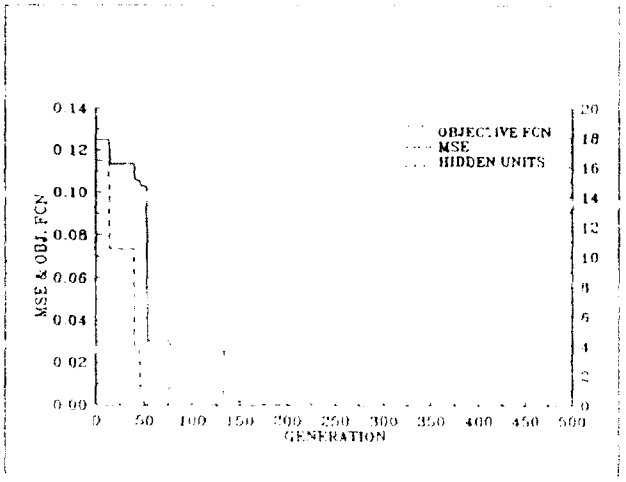

Figure 9. Evolving hidden units for the XOR mapping. The search was conducted over 20 possible hidden units.

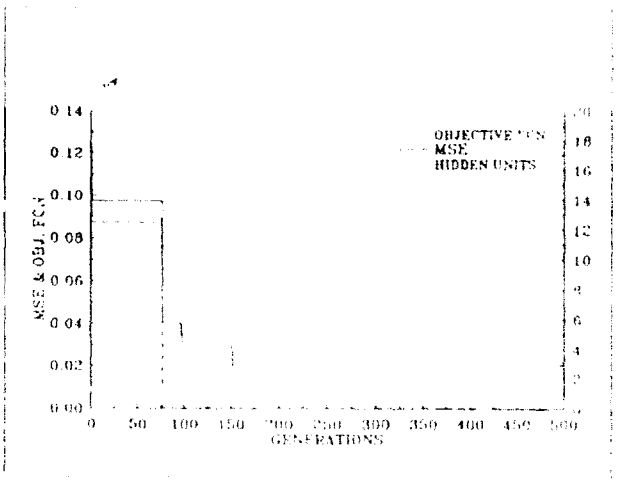

Figure 10. Evolving hidden units for the XOR mapping. The search was conducted over 50 possible hidden units.

5. EVOLVING STRUCTURE: A SELF-ORGANIZING NETWORK

A nicety of using the EP search technique is that additional terms can be incorporated in the objective function without requiring a major programming modification. An example of this feature is the multi-dimensional path planning work done by Page *et al.*[18]. Wah and Kriplani[19] have incorporated training time into their objective function

$$J = A - B \times Training\ Time - C \times Cost$$

where A, B and C are constants. The *Cost* term is a function of the network size and number of weights and is given by

$$Cost = N_{neurons} + 0.1\sqrt{N_{weights}}$$

An investigation very similar to this work has been conducted by Bornholdt and Graudenz[20]. A major difference between the methods is that Bornholdt and Graudenz used genetic algorithms for evolving the network structure. Further, the networks implemented in the referenced work can have asynchronous updates and recurrent structure in the hidden layer. Bornholdt and Graudenz define a dilution ratio $D = N_{synapses}/N^2_{neurons}$. For example, the dilution ratio for the network shown in Fig. 8 is $D = 0.33$. This result does not have the small dilution ratios found using recurrent hidden structure. For the simple XOR mapping, it does not take as many generations to train since the stability issues associated with recurrent structure does not effect the system.

Currently, the feed-forward structure is maintained and the objective function is modified to incorporate both cost associated with the number of connections and number of neurons

$$J = \alpha E + \beta N_{connections} + \gamma N_{nodes}$$

Initially, the weighting coefficients were kept the same: $\alpha=1$, $\beta=0.0001$, $\gamma=0.01$. The methods used in sections 3 and 4 were combined and tested on the XOR mapping. Fig. 11 shows the training results for a hidden layer domain of 50 hidden units. After 2500 generations, the network has evolved to a 2-2-1 structure. However, the mean sum-squared pattern error has not been sufficiently reduced as shown in Fig. 11(b). This is due to the manner in which offspring are generated. Each new offspring contains a new structure as well as perturbed weight set. Chances are, very little weight adaptation is accomplished for any given structure.

The adaptation problem became more apparent when the same approach was taken with the 3-bit parity mapping. To overcome this limitation, two offspring are generated for each parent. The first offspring is generated with a perturbed weight set and new structure. The second offspring is generated with a perturbed weight set and the same structure as its parent. Results for the 3-bit parity mapping problem are shown in Fig. 12. This experiment was conducted with weighting coefficients of $\alpha=1$, $\beta=0.0001$, $\gamma=0.001$. The hidden layer domain consisted of 25 hidden units. After 500 generations, the mean sum-squared pattern error has been sufficiently reduced for 11 hidden units as shown in Fig. 12(b). Further optimization occurs by reducing the number of connections as shown in Fig. 12(c). The final configuration is shown in Fig. 13. Obviously, not all of the remaining hidden units are connected and the bias redundancy can be eliminated to further reduce the node count. The statistics at the end of 5000 generations are $N_{connections} = 18$, $N_{nodes} = 11$, $E = 0.000052$, $J = 0.012852$ and a dilution ratio of $D = 0.15$.

This modified approach was applied to the XOR mapping problem with 10 parent networks. For networks that were fully connected from the onset, this search technique quickly found a fully-connected 2-2-1 structure and weight adaptation occurred over subsequent generations. To make the problem more challenging, each network was instantiated with a 50% probability of any given connection being present. The resulting structure is shown in Fig. 14 and the statistical data is shown in Fig. 15. This experiment was conducted with weighting coefficients of $\alpha = 1$, $\beta = 0.0001$, $\gamma = 0.001$. The hidden layer domain consisted of 50 hidden units. The statistics at the end of 1000 generations are $N_{connections} = 13$, $N_{nodes} = 11$, $E = 0.000000$, $J = 0.012300$ and a dilution ratio of $D = 0.11$.

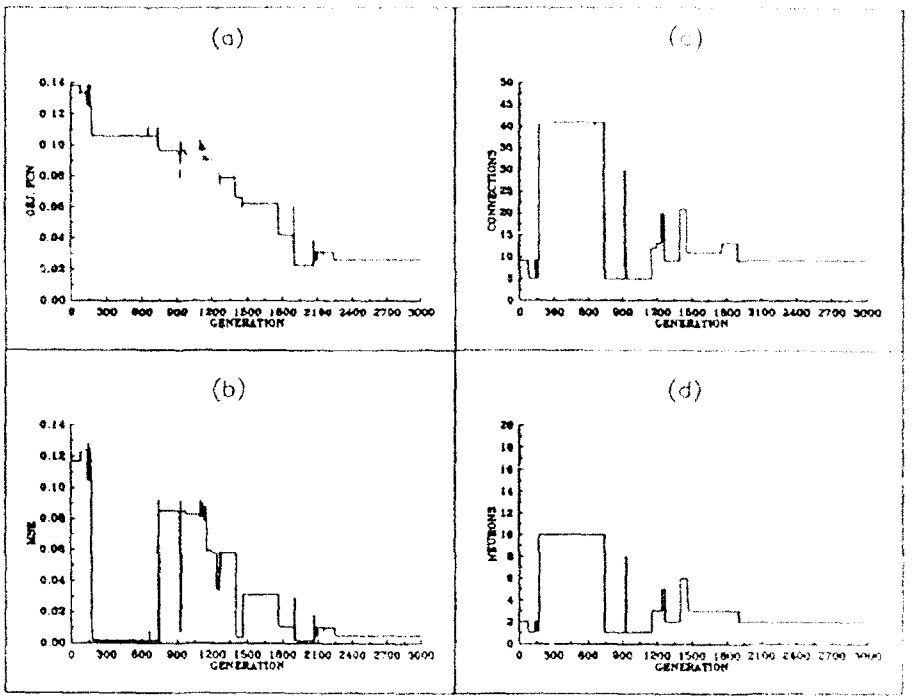

Figure 11. Evolving weights and structure for the XOR mapping: (a) cost function, (b) mean sum-squared pattern error, (c) number of connections, and (d) the number of neurons.

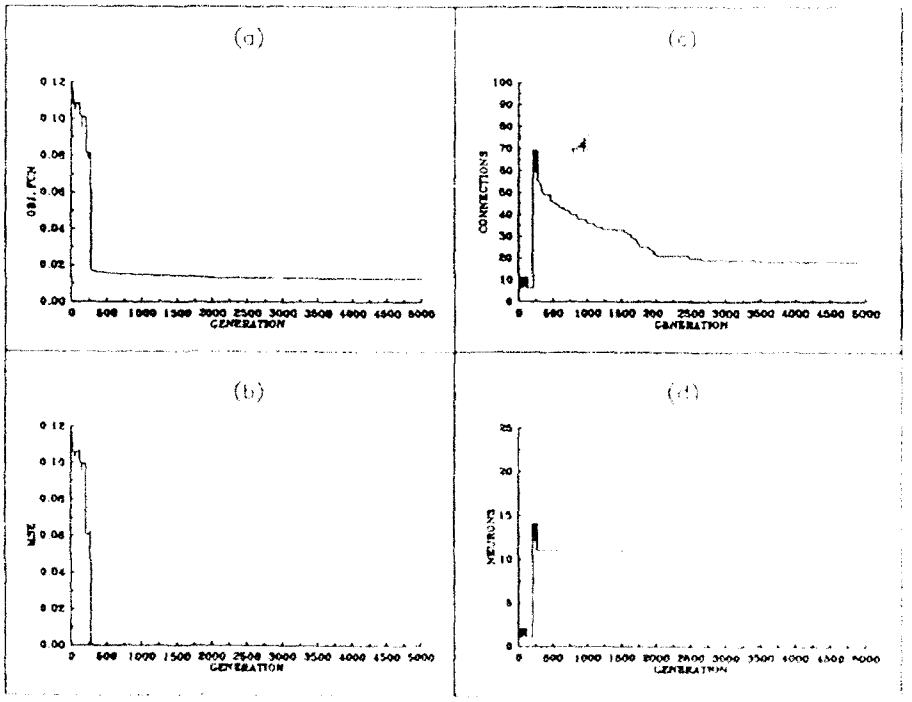

Figure 12. Evolving weights and structure for 3-bit parity: (a) objective function, (b) the mean sum-squared pattern error, (c) number of connections and (d) number of neurons.

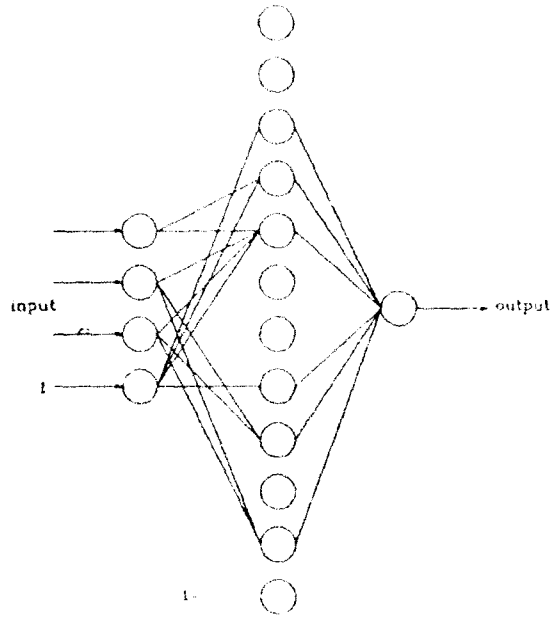

Figure 13. The final evolved architecture for the 3-bit parity mapping problem.

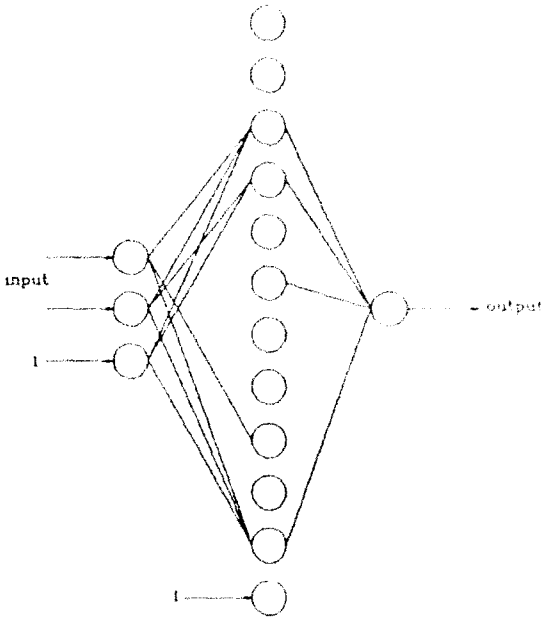

Figure 14. The final evolved architecture for the XOR mapping problem.

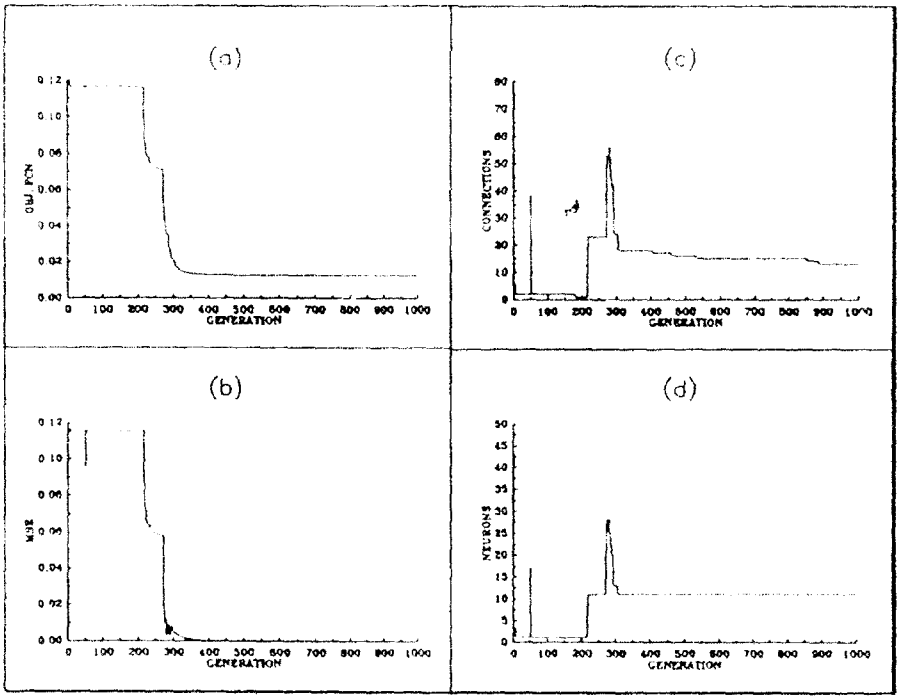

Figure 15. Evolving weights and structure for the XOR mapping: (a) cost function, (b) mean sum-squared pattern error, (c) number of connections, and (d) the number of neurons.

6. CONCLUSIONS

Experiments have shown that evolutionary programming can be used for simultaneously determining both network architecture and parameters. These results indicate that the EP search paradigm has high potential for automating the design of neural networks. It has been demonstrated that the general architecture shown in Fig. 1 can be achieved for simple binary mapping problems. However, some applications may be hindered by the excess memory requirements needed by multi-agent searches[12] or the training time necessary to obtain adequate convergence. If the design process for a specific problem is not capable of being fully automated, then this technique may prove useful in providing a designer with insight to the task at hand as well as a reasonable starting point for further investigations.

Caution should be used in trying to obtain minimal size networks. As Sietsma and Dow[7,8] point out, pruning networks reduces their generalization capabilities if noise is present. Experience has that shown that adding a slight amount of noise to the training data will result in more robust classifiers. Since only binary mapping problems were investigated, it is not clear how the approach given in this study will work on classification or continuous mapping problems. This remains an area of further research.

Nevertheless, stochastic training techniques are becoming prevalent in neurocomputing (especially in hardware implementations[21]). Further research should allow for a more general connectivity structure including interconnections among the hidden units as well as connections between input and output neurons. Since model structure is continually being modified, it is incumbent that an information criteria I be addressed either explicitly or implicit in the objective function (*e. g.*)

$$J = \alpha E + \beta N_{connections} + \gamma N_{nodes} + \zeta f(I)$$

providing a viable approach for automated model selection based not only on error and computation cost, but on the information content as well.

REFERENCES

1. L. J. Fogel, A. J. Owens and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley and Sons, 1966.

2. D. B. Fogel, "An evolutionary approach to the traveling salesman problem", *Biological Cybernetics*, Vol. 60, No. 2, 1988.

3. D.B. Fogel, L.J. Fogel and V.W. Porto, "Evolving neural networks", *Biological Cybernetics*, Vol. 63, 1990.

4. D. B. Fogel, *System Identification through Simulated Evolution: a Machine Learning Approach to Modeling*, Ginn Press, Needham, MA., 1991.

5. T. Ash, "Dynamic node creation in backpropagation networks", Int. Joint Conf. on Neural Networks, San Diego, 1989.

6. Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units", *Neural Networks*, Vol. 4, 1991.

7. J. Sietsma and R. J. F. Dow, "Neural net pruning: why and how", IEEE Int. Conf. on Neural Networks, San Diego, 1988.

8. J. Sietsma and R.J.F. Dow, "Creating artificial neural networks that generalize", *Neural Networks*, Vol. 4, 1991

9. S. Li, "An optimized backpropagation algorithm with minimum norm weights", Int. Joint Conf. on Neural Networks, San Diego, 1990.

10. A.W. Bailey, "Automatic evolution of neural net architectures", Int. Joint Conf. on Neural Networks, Washington, D.C., 1990.

11. M.F. Tenorio and W.T. Lee, "Self-organizing network for optimum supervised learning", *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, March 1990.

12. J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, 1991.

13. E. Aârts and J. Korst, *Simulated Annealing and Boltzman machines: a Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley & Sons, 1989.

14. D. H. Ackley, "Stochastic iterated genetic hill-climbing", CMU-CS-87-107, Ph. D. dissertation, Computer Science Dept., Carnegie Mellon University, Pittsburgh, PA, 1987.

15. R. Winter and B. Widrow, "Madaline rule II: a training algorithm for neural networks", Int. Joint Conf. on Neural Networks, 1988.

16. D. Waagen, P. Diercks, and J. R. McDonnell, "The stochastic direction set algorithm: a hybrid technique for finding function extrema", First Annual Conf. on Evolutionary Programming, San Diego, 1992.

17. A.S. Weigend, D.E. Rumelhart, and B.A. Huberman, "Back-propagation, weight elimination and time series prediction", *Connectionist Models: Proceedings of the 1990 Summer School*, D. S. Touretzky [ed.], Morgan Kaufman, 1991.

18. W. C. Page, J. R. McDonnell, and B. L. Andersen, "An evolutionary programming approach to multi-dimensional path planning", First Annual Conf. on Evolutionary Programming, San Diego, 1992.

19. B. W. Wah and H. Kriplani, "Resource constrained design of artificial neural networks", Int. Joint Conf. on Neural Networks, San Diego, 1990.

20. S. Bornholdt and D. Graudenz, General asymmetric neural networks and structure design by genetic algorithms, DESY 91-046, ISSN 0418-9833, Deutches Elektonen-Synchrotron, Hamburg, May 1991.

21. B. W. Lee and B. J. Shen, "Analysis and design of analog VLSI neural networks", in *Neural Networks for Signal Processing*, B. Kosko (Ed.), Prentice-Hall, 1992.